

COMPTE RENDU

TP : JPA Hibernate Spring Data - Many To Many

Par : Assimi DIALLO

Encadrant : M. YOUSSEFI

JPA Hibernate Spring Data - Many To Many

Dans ce TP nous allons voir comment se font les relations Many To Many avec JPA Hibernate et Spring Data. La particularité des relations Many To Many c'est la création de la table d'association résultante des 2 autres.

Nous allons créer une table **Users** et une table **Role**. Entre ces 2 tables on a une relation Many To Many car un User peut avoir plusieurs rôles et un Role peut correspondre à plusieurs users.

Création des entités : On crée 2 classes User et Role. Chacune des 2 classes prend comme attribut une liste de l'autre classe. Juste au dessus de chaque liste on a l'annotation **@ManyToMany** et c'est ce qui rend effective la relation.

```

15  @Entity
16  @Data @NoArgsConstructor @AllArgsConstructor
17  public class User {
18      @Id
19      private String id;
20      private String userName;
21      private String password;
22      @ManyToMany(mappedBy = "users", fetch = FetchType.EAGER)
23      @ToString.Exclude
24      private List<Role> roles = new ArrayList<>();
25  }

```

```

12  @Entity
13  @Data @NoArgsConstructor @AllArgsConstructor
14  public class Role {
15      @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
16      private Long id;
17      @Column(name = "description")
18      private String desc;
19      @Column(unique = true)
20      private String roleName;
21      @ManyToMany(fetch = FetchType.EAGER)
22      @ToString.Exclude
23      private List<User> users = new ArrayList<>();
24  }

```

Quand on lance l'application on obtient dans notre base de données
H2 :

```

jdbc:h2:mem:users_db
+ [ ] ROLE
+ [ ] ROLE_USERS
+ [ ] USER
+ [ ] INFORMATION_SCHEMA
+ [ ] Sequences
+ [ ] Users
(i) H2 1.4.200 (2019-10-14)

```

On remarque que les 2 tables ont été créées. Mais on remarque aussi la table d'association : **ROLE_USERS**.

On crée une interface **UserService** dans un package **service**.
L'implémentation **UserServiceImpl** de l'interface **UserService** va nous permettre d'interagir avec la base de données. On y utilise **userRepository** et **roleRepository** des interfaces **UserRepository** et **RoleRepository**, qui héritent de **JpaRepository**, et qui sont dans le package **repositories** :

```
UserService.java
1 package enset.ma.jpamanymany.service;
2
3 import enset.ma.jpamanymany.entities.Role;
4 import enset.ma.jpamanymany.entities.User;
5
6 public interface UserService {
7     User addNewUser(User user);
8     Role addNewewRole(Role role);
9     User findUserByUsername(String username);
10    Role findRoleByRoleName(String role);
11    void addRoleToUser(String user, String role);
12    User authenticate(String username, String password);
13 }
14
```

```
UserServiceImpl.java
1 package enset.ma.jpamanymany.service;
2
3 import ...
4
13
14 @Service
15 @Transactional
16 @AllArgsConstructor
17 public class UserServiceImpl implements UserService {
18     private UserRepository userRepository;
19     private RoleRepository roleRepository;
20
21     @Override
22     public User addNewUser(User user) {
23         user.setId(UUID.randomUUID().toString());
24         return userRepository.save(user);
25     }
26
27     @Override
28     public Role addNewewRole(Role role) {
29         return roleRepository.save(role);
30     }
31
32     @Override
33     public User findUserByUsername(String username) { return userRepository.findUserByName(username); }
34
36
```

```

37     @Override
38     public Role findRoleByRoleName(String role) { return roleRepository.findRoleByRoleName(role); }
41
42     @Override
43     public void addRoleToUser(String username, String rolename) {
44         User user = userRepository.findUserByUserName(username);
45         Role role = roleRepository.findRoleByRoleName(rolename);
46         if(user.getRoles()!=null){
47             user.getRoles().add(role);
48             role.getUsers().add(user);
49         }
50         userRepository.save(user);
51     }
52
53
54     @Override
55     public User authenticate(String username, String password) {
56         User user = userRepository.findUserByUserName(username);
57         if(user==null) throw new RuntimeException("Bad credentials");
58         if(user.getPassword().equals(password)){
59             return user;
60         }
61         throw new RuntimeException("Bad credentials");
62     }
63 }
64

```

On peut donc faire les différentes opérations de remplissage de la base de données depuis **JpamanymanyApplication** :

```

1  package enset.ma.jpamanymany;
2
3  import ...
14
15  @SpringBootApplication
16  public class JpamanymanyApplication {
17
18      public static void main(String[] args) { SpringApplication.run(JpamanymanyApplication.class, args); }
21
22      @Bean
23      CommandLineRunner start(UserService userService){
24          return args ->{
25              Stream.of("assimdll", "diallo")
26                  .forEach(username->{
27                      User user = new User();
28                      user.setUserName(username);
29                      user.setPassword("12345");
30                      userService.addNewUser(user);
31                  });
32
33              Stream.of("STUDENT", "USER", "ADMIN")
34                  .forEach(rolename->{
35                      Role role = new Role();
36                      role.setRoleName(rolename);
37                      userService.addNewewRole(role);
38                  });

```

```

40     userService.addRoleToUser( user: "assimdll", role: "STUDENT");
41     userService.addRoleToUser( user: "assimdll", role: "USER");
42     userService.addRoleToUser( user: "diallo", role: "USER");
43     userService.addRoleToUser( user: "diallo", role: "ADMIN");
44
45     try {
46         User user = userService.authenticate( username: "assimdll", password: "12345");
47         //System.out.println(user.getId());
48         System.out.println(user.getUserName());
49         System.out.println("Roles----->");
50
51         user.getRoles().forEach(r->{
52             System.out.println("role : "+r.toString());
53         });
54
55     }catch(Exception e) {
56         e.printStackTrace();
57     }
58 };
59
60
61
62 }
63
64 }

```

Quand on exécute **JpamanymanyApplication**, on obtient :

Table USER :

SELECT * FROM USER;

| ID | PASSWORD | USER_NAME |
|--------------------------------------|----------|-----------|
| adcbdd30-ee49-4396-ae82-5ecf54323a9a | 12345 | assimdll |
| d1bd1cf2-df86-4c76-b07f-99d18a7ddc8d | 12345 | diallo |

(2 rows, 9 ms)

Edit

Table ROLE :

SELECT * FROM ROLE;

| ID | DESCRIPTION | ROLE_NAME |
|----|-------------|-----------|
| 1 | null | STUDENT |
| 2 | null | USER |
| 3 | null | ADMIN |

(3 rows, 12 ms)

Edit

Table ROLE_USERS

```
SELECT * FROM ROLE_USERS;
```

| ROLES_ID | USERS_ID |
|----------|--------------------------------------|
| 1 | adcbdd30-ee49-4396-ae82-5ecf54323a9a |
| 2 | adcbdd30-ee49-4396-ae82-5ecf54323a9a |
| 2 | d1bd1cf2-df86-4c76-b07f-99d18a7ddc8d |
| 3 | d1bd1cf2-df86-4c76-b07f-99d18a7ddc8d |

(4 rows, 0 ms)

On peut vérifier aussi que la fonction `authenticate` définie tout en bas. On nous affiche l'utilisateur et ses rôles quand le mot de passe et le `username` sont corrects sinon on affiche **Bad Credentials**.

```
assimdll
Roles----->
role : Role(id=1, desc=null, roleName=STUDENT)
role : Role(id=2, desc=null, roleName=USER)
|
```

Maintenant modifions le mot de passe passé en paramètre et mettons un mot de passe erroné :

```
try {
    User user = userService.authenticate( username: "assimdll", password: "00000");
    //System.out.println(user.getUserId());
    System.out.println(user.getUserName());
    System.out.println("Roles----->");

    user.getRoles().forEach(r->{
        System.out.println("role : "+r.toString());
    });
}catch(Exception e) {
    e.printStackTrace();
}
```

L'exécution nous donne cette fois ci :

```
java.lang.RuntimeException: Create breakpoint : Bad credentials
    at enset.ma.jpamanymany.service.UserServiceImpl.authenticate(UserServiceImpl.java:61)
    at enset.ma.jpamanymany.service.UserServiceImpl$$FastClassBySpringCGLIB$$66399897.invoke(<generated>)
    at org.springframework.cglib.proxy.MethodProxy.invoke(MethodProxy.java:218)
    at org.springframework.aop.framework.CglibAopProxy$CglibMethodInvocation.invokeJoinpoint(CglibAopProxy.java:783)
```

On peut maintenant déplacer notre base de données de H2 vers MySQL :

```
application.properties
1 #spring.datasource.url=jdbc:h2:mem:users_db
2 #spring.h2.console.enabled=true
3 server.port=8086
4 spring.datasource.url=jdbc:mysql://localhost:3306/users-db?createDatabaseIfNotExist=true
5 spring.datasource.username=root
6 spring.datasource.password=
7 spring.jpa.hibernate.ddl-auto=update
8 spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MariaDBDialect
9
```

```
<!--<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId>
  <scope>runtime</scope>
</dependency>-->
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
</dependency>
```

Quand on exécute :

| Table | Action | Lignes | Type | Interclassement | Taille | Perte |
|-------------------------------------|--|--------|--------|--------------------|----------|-------|
| <input type="checkbox"/> role | ★ Parcourir Structure Rechercher Insérer Vider Supprimer | 3 | InnoDB | utf8mb4_general_ci | 32,0 Kio | - |
| <input type="checkbox"/> role_users | ★ Parcourir Structure Rechercher Insérer Vider Supprimer | 4 | InnoDB | utf8mb4_general_ci | 48,0 Kio | - |
| <input type="checkbox"/> user | ★ Parcourir Structure Rechercher Insérer Vider Supprimer | 2 | InnoDB | utf8mb4_general_ci | 16,0 Kio | - |
| 3 tables Somme | | 9 | InnoDB | utf8mb4_general_ci | 96,0 Kio | 0 o |

☐ Tout cocher Avec la sélection :

+ Options

| | | id | password | user_name |
|--------------------------|----------|--------------------------------------|----------|-----------|
| <input type="checkbox"/> | ✎ Éditer | 0326170d-0743-4d4b-8ceb-153e68a78e7d | 12345 | diallo |
| <input type="checkbox"/> | ✎ Éditer | 8b2878c3-fab3-4534-a0ca-a6aafcc8ae65 | 12345 | assimdl |

☐ Tout cocher Avec la sélection :

✎ Éditer Copier Supprimer Exporter

+ Options

| | | | | | id | description | role_name |
|--------------------------|--|--------|--|--------|----|-------------|----------------|
| <input type="checkbox"/> | | Éditer | | Copier | | Supprimer | 1 NULL STUDENT |
| <input type="checkbox"/> | | Éditer | | Copier | | Supprimer | 2 NULL USER |
| <input type="checkbox"/> | | Éditer | | Copier | | Supprimer | 3 NULL ADMIN |

☐ Tout cocher
 Avec la sélection : Éditer Copier

+ Options

| roles_id | users_id |
|----------|--------------------------------------|
| 1 | 8b2878c3-fab3-4534-a0ca-a6aafcc8ae65 |
| 2 | 8b2878c3-fab3-4534-a0ca-a6aafcc8ae65 |
| 2 | 0326170d-0743-4d4b-8ceb-153e68a78e7d |
| 3 | 0326170d-0743-4d4b-8ceb-153e68a78e7d |

☐ Tout afficher | Nombre de lignes : 25

La prochaine étape sera de mettre en place un contrôleur pour retourner des données sous format JSON en fonction d'un paramètre passé dans l'url. Ce contrôleur sera défini dans un package **web** :

```
UserController.java
3 import enset.ma.jpamanymany.entities.User;
4 import enset.ma.jpamanymany.service.UserService;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.web.bind.annotation.GetMapping;
7 import org.springframework.web.bind.annotation.PathVariable;
8 import org.springframework.web.bind.annotation.RestController;
9
10 @RestController
11 public class UserController {
12     @Autowired
13     private UserService userService;
14     @GetMapping("/users/{username}")
15     public User user(@PathVariable String username){
16         User user = userService.findUserByUsername(username);
17         return user;
18     }
19 }
```


On obtient sur l'adresse <http://localhost:8086/users/assimdll> :

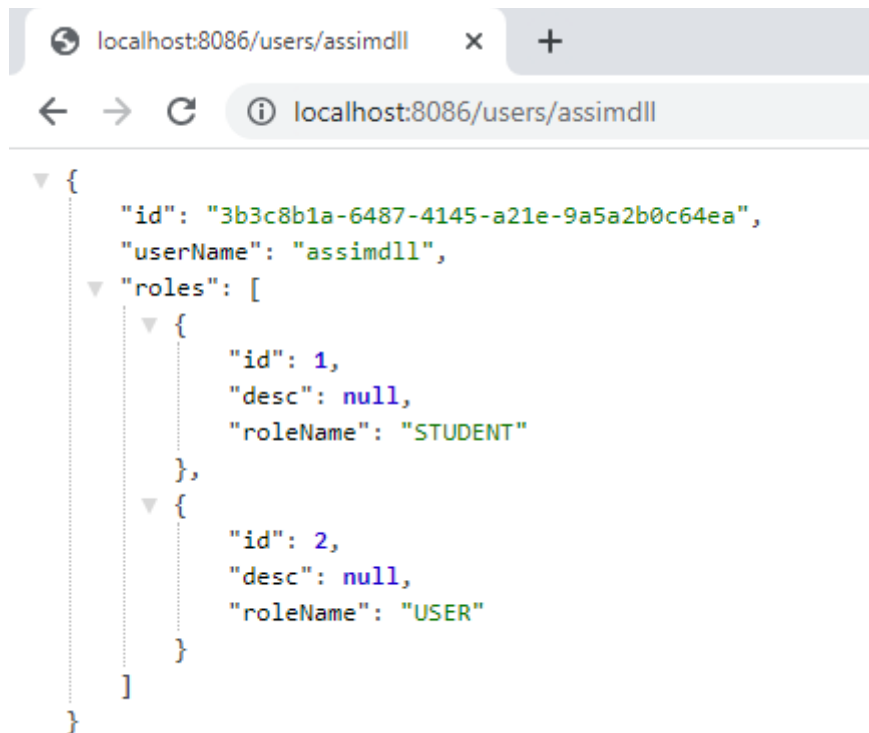


Il s'agit d'une boucle infinie car on retourne un utilisateur qui vient avec un rôle qui retourne l'utilisateur qui retourne le rôle et ainsi de suite. Pour y remédier on doit ajouter une propriété qui va empêcher le rôle d'afficher ou de retourner l'utilisateur ou les utilisateurs correspondant. On va profiter avec la même propriété empêcher l'affichage du mot de passe de l'utilisateur. On aura dans les classes User et Role :

```
15 @Entity
16 @Data @NoArgsConstructor @AllArgsConstructor
17 public class User {
18     @Id
19     private String id;
20     private String userName;
21     @JsonProperty(access = JsonProperty.Access.WRITE_ONLY) //Pour ne pas retourner le mdp dans JSON
22     private String password;
23     @ManyToMany(mappedBy = "users", fetch = FetchType.EAGER)
24     @ToString.Exclude
25     private List<Role> roles = new ArrayList<>();
26 }
```

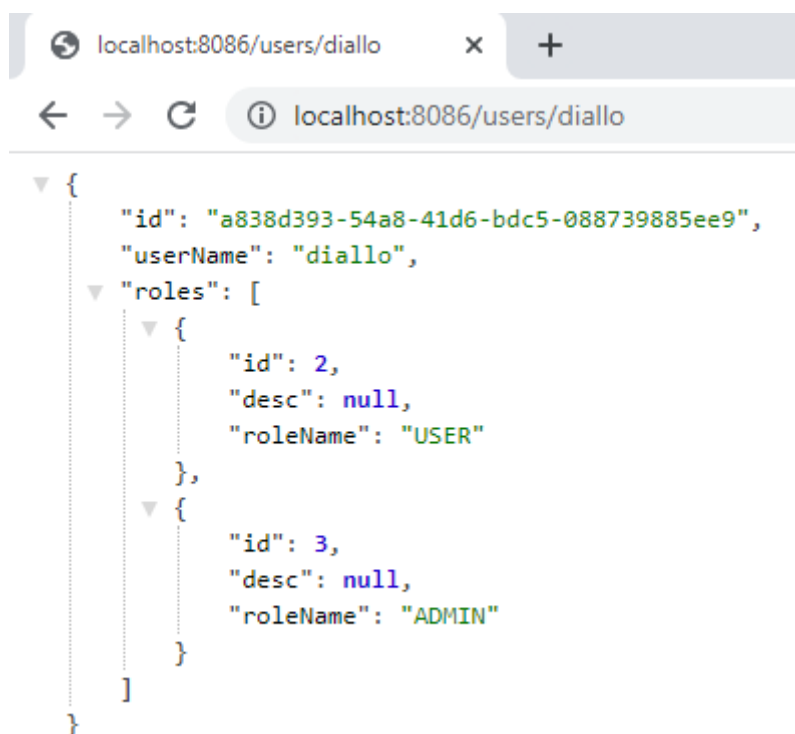
```
12 @Entity
13 @Data @NoArgsConstructor @AllArgsConstructor
14 public class Role {
15     @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
16     private Long id;
17     @Column(name = "description")
18     private String desc;
19     @Column(unique = true)
20     private String roleName;
21     @ManyToMany(fetch = FetchType.EAGER)
22     @ToString.Exclude
23     @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
24     List<User> users = new ArrayList<>();
25 }
```

Quand on exécute maintenant on obtient :



```
{
  "id": "3b3c8b1a-6487-4145-a21e-9a5a2b0c64ea",
  "userName": "assimd11",
  "roles": [
    {
      "id": 1,
      "desc": null,
      "roleName": "STUDENT"
    },
    {
      "id": 2,
      "desc": null,
      "roleName": "USER"
    }
  ]
}
```

En allant sur l'adresse : <http://localhost:8086/users/diallo>



```
{
  "id": "a838d393-54a8-41d6-bdc5-088739885ee9",
  "userName": "diallo",
  "roles": [
    {
      "id": 2,
      "desc": null,
      "roleName": "USER"
    },
    {
      "id": 3,
      "desc": null,
      "roleName": "ADMIN"
    }
  ]
}
```

C'est ainsi que se termine ce TP sur JPA, Hibernate et Spring Data pour les relations Many To Many.